cMix Tagging Attack

Analysis and Mitigation

Benjamin Wenger

Attack Description and Previous Work

As described in section 5.2 of the cMix academic paper[1], a node can perform a tagging attack by multiplying in an uncommitted factor to a message and then seeing which message comes out correctly when inverting the factor.

Furthermore, as described in section 5.2, with cMix in its base construction, it is possible for the last node to execute this attack and undo the inversion before delivering the messages, doing so without a trace. In the paper, they propose delaying the final unwrapping stage of cmix (from this point on referred to as group open) until after the last node provides a commitment on the final outputs in order to ensure the attack cannot be carried out. Group Open does not ensure the tagging attack cannot successfully break privacy, but does ensure if the last node in the team is part of the attack they are unable to make the attack undetectable. Further optimizations on this solution can be found in the Appendix.

In the paper, the primary solution presented is in sections 5.1, 5.2, and 5.3. The solution can be summarized as an interactive commitment scheme and an opening verification process for those commitments when faults are detected, which allows a third party to conclusively determine which node has executed the attack. This solution does not ensure the attack cannot be executed, but is able to create an effective deterrent to the attack. It is the gold standard solution but is extremely difficult to implement due to its interaction with clients, and consensus and the blockchain to enforce requests in a byzantine safe manner.

Alternative solution: Indistinguishability of Outputs

A fundamental facet of the problem is the requirement that the attacker can distinguish when messages are output correctly. If a valid message is indistinguishable from an invalid message, then it will be impossible to detect when

[1] https://eprint.iacr.org/2016/008.pdf

the inversion of the invalid factor un-corrupts a message. Given that message contents are end to end encrypted, the sole identifying component of outputs messages is the User ID.

A solution therefore could be to have ephemeral, constantly changing IDs. If identities are never reused, then they cannot be used to detect proper decryption. Such solutions are indeed possible, but create significant overhead for message pickup.

Indistinguishability of outputs can be achieved through another mechanism - through forced ID collision. By creating reception IDs in a dense space where the number of available IDs is roughly on par with the number of IDs in use, collisions between random valid IDs and known valid IDs can be made to not just be common, but overwhelmingly likely. As a result, on executing a tagging attack it will be likely that attackers will be unable to identify their tagged message.

Adversarial Model

- It is assumed that only the attacker is capable of executing the attack when they are a member of a team and only for messages which are processed by that team
- It is assumed that the attacker knows the recipients addresses for all received messages in all rounds. [2]
- It is assumed that the attacker knows all sender ID <-> IP address relationships for each round and all recipient ID <-> IP address relationships each round. This can be derived from the global adversarial model or the standard BFT model used by the consensus portion of the network
- Clients have private keys and data which is not known by the Adversary and can share this data between each other securely

Solution

The solution is to restrict the address space in which messages are received within the network.

Assuming a number of active recipients *R*, the number of addresses the network maintains available (*A*) will be set at:

$$A = 2^s \quad s = floor(log_2 R)$$

[2] This can be gleaned from gossips used to disseminate data for rate limiting account and delivering recipient data to allow a client to learn where they have messages available from any gateway

This will ensure the number of available addresses will always be fewer than the number of recipients, ensuring a high likelihood of collisions which obfuscate tagged messages.

This equation is chosen because it heavily simplifies the address creation process to have the space bit aligned (an example is described in the appendix). The floor is used (instead of a ceiling) because the system operates more securely when the number of addresses is less than R, rather than greater.

Analysis

These addresses will not be statically assigned but will rotate at a set period p. Rotation will be at a random phase only known by clients participating in the sending and receiving. As a result, it will not be possible to get an exact list of currently active addresses, and an attacker will be required to build a list of potentially active recipient addresses. This will correlate to the number of addresses selected by recipients within the set (R_p)

$$R_p = 2R$$

When an attacker adds their factor, the resulting address can be considered to be randomly chosen. As a result, the probability of a collision is simply the probability that at least one other recipient selected the address.

Due to the fact that selection of address is effectively random, the probability that a single recipient selects the address is:

$$p_{selection\ single} = 1/A$$

And the probability that the address is not selected is:

$$p_{no\ selection\ single} = 1 - p_{selection\ single} = 1 - 1/A$$

Given that the probability of at least one selection is the inverse of the probability of no selection, it can be computed that the probability of selection as follows:

$$p_{selection} = 1 - p_{no\ selection} = 1 - \left(p_{no\ selection\ single}\right)^{selections} = 1 - (1 - 1/A)^{R_}$$

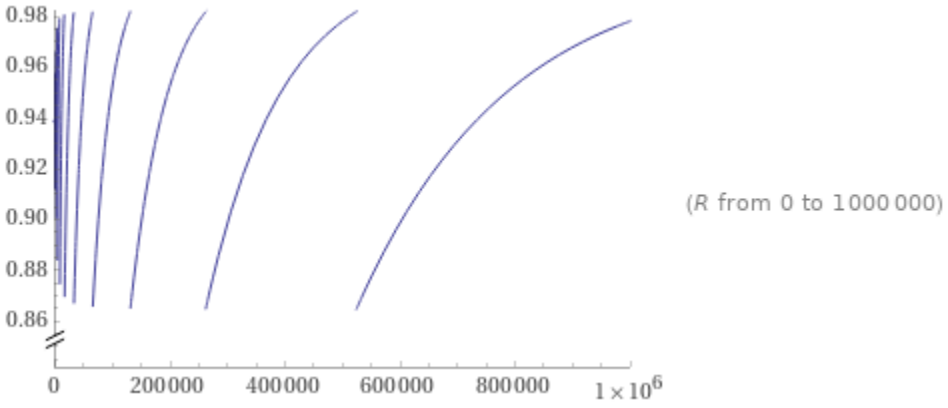
Which can be simplified into terms of R

$$p_{selection} = 1 - (1 - 1/\left(2^{floor(log_2 R)}\right))^{2R}$$

From this equation it can be seen that it only depends on R.

Calculation

By graphing the output, it can be seen that the probability repeats every $2^n < R < 2^{n+1}$.



Therefore we can just analyze a single region to understand the probabilities. The probabilities in a single region can be calculated with the the following python code:

```
import math

//check values between 2^10 and 2^11
i = 1025

while i<2047:
    a = 2**math.floor(math.log(i)/math.log(2))
    pSingle = 1-1/a
    pAdv = 1-pow(pSingle,2*i)
    print(f'{i}, {pAdv}')
    i= i+10
```

This shows that the collision rate is between 86.5% and 98.2%, meaning that when a tagging attack is completed, the attacker is unlikely to be able to detect the message they tagged.

Group Open

This approach provides a defense in depth that is complementary to the Group Open. While group open prevents the attack, ID collisions prevent the attacker from even finding their tagged message with enough certainty to invert the factor without having a greater chance of simply corrupting more messages. While this does not remove the potential need for group agreement on the output for Praxxis consensus it minimizes the impact of a new and novel tagging attack that bypasses group open.

Analysis of a Tagging Attack in this Environment

WIth this mitigation in place, a tagging attack can still occur, but it becomes much more difficult. In general the attacker's approach will be to execute the tagging attack many-many times and use the imperfect probabilities it creates to track the IP addresses which are associated with IDs found until patterns emerge.

Essentially, the attacker will have to record the address found when removing the factor from every message and track if any IP addresses are associated with them. Given the high collision rate and the regular ID rotation, they will find most addresses in the set have IP addresses associated and will have to execute the attack many times to find IP addresses which are more common than those found by random chance. They may be able to boost the success of this attack by giving more priority to IPs from converted addresses where the initial address was unknown.

Client Impact of Address Collisions

Having addresses in a small space impacts message pickup because you can have users using the same Identity. As long as the number of collisions is not too high, the extra bandwidth is not significant and the collisions add IP address ambiguity, further hampering adversarial attacks.

To model the average number of collisions on an address, one would find the mean of the binomial distribution of collisions for that address. This would be the mean of a binomial distribution:

$$p_{collision} = P(x; n = R, p = 1/A)$$

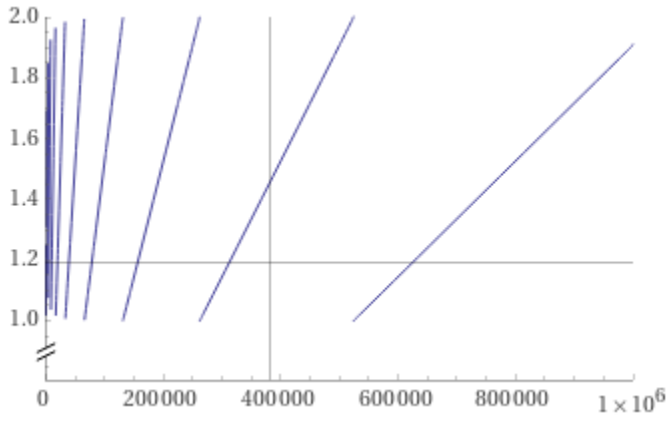
Given that the equation for the mean of a binomial distribution is:

$$Mean = np$$

The average number of collisions can be calculated as:

$$Avg\ Collisions = R/A = R/2^{floor(log_2 R)}$$

Like the rate of collisions between users, this will be a repetitive and repeats every $2^n < R < 2^{n+1}$



Therefore we can just analyze a single region to understand the probabilities. The probabilities in a single region can be calculated with the the following python code:

```
import math

//check values between 2^10 and 2^11
i = 1025

while i<2047:
    a = 2**math.floor(math.log(i)/math.log(2))
    mean = i/a
    print(f'{i}, {mean}')
```

```
    i= i+10
```

From this we can see that the average number of users using a lot varies between 1 and 2.

It is also important to understand how frequently many more collisions will take place. The standard deviation of a binomial distribution can be calculated as:

$$Standard\ Deviation = \sqrt{np(1-p)} = \sqrt{R2^{-floor(log_2 R)}(1-2^{-floor(log_2 R)})}$$

```
import math

//check values between 2^10 and 2^11
i = 1025

while i<2047:
    a = 2**math.floor(math.log(i)/math.log(2))
    stDev = math.sqrt(i* 1/a * (1-1/a))
    print(f'{i}, {stDev}')
    i= i+10
```

This leads to standard deviations ranging from 1 to 1.4. Overall, this means that collisions with a very large number of clients will be rare.

Selecting R

Due to the obfuscation through collisions this approach adds to the network, it is not possible to directly measure R. Due to gossips, it is possible to measure a different related value, Addresses utilized (A_U). This is a measurement of how many addresses are utilized by the network over a given period of time. In a simple approach where all addresses rotate together, the A_U can be approximated via:

$$A_U \simeq R \times Mean = R \times R/A$$

Where A is the current address size

Using simple algebra, we can then calculate R:

$$R \simeq \sqrt{A_U A}$$

The problem with this result is that clients do not all rotate their reception identities at a set time. Assuming identities rotate at a period *p*, and usage rates are constant across the period, it can be found that only 75% of identities are valid[3], with 25 percent being duplicates, making the equation:

$$R \simeq \sqrt{0.75 A_U A}$$

Varying R

The number of active Recipients on the network (R) will be constantly changing dependent on usage. It is impossible to constantly vary the address space A as R varies. As a result, two conclusions can be drawn:

- The period *p* must be selected such that the utilization of the server does not vary by more than a factor of 2 between subsequent periods. A recommended *p* is 24 hours.
- The rate that the address space changes must be much much greater than *p*

A mechanism to determine the number of true recipients has not been designed yet, but will be dependent on heuristic properties of platform usage. A heuristic approach could be based upon the rate of message collision across a sample of clients.

Modifying the cMix packet

The cMix packet always has the 1st bit, and the 4097th bit as 0 in order to ensure the 2 sub payloads are within the cyclic group. This means that if you randomly mutate the payloads (as you do in a tagging attack) there is only a 25% chance both will come out as 0, making it indistinguishable from a well formed payload. This needs to be fixed by modifying the spec to make those 2 bits 1 or 0.

[3] This can be found by integrating over the period the percentage of the use of IDs which are unique to the period

Implementation

Network Identities

Within the cmix system as implemented, there is a defined ID structure:

Value (256 bits)	Type (8 bits)

The types are as follows:

Type	Number	Description
Generic	0x00	Components which do not fit with other classifications. Example: Permissioning Server
Node	0x01	Nodes within the network
Gateway	0x02	Gateways within the network. Will always have the same value as its associated node
User	0x03	A user in the network

The value component of the ID is the blake2B hash of an RSA public key and a 256 bit salt

$$value = hash(pubKey, salt)$$

This allows IDs to be independently generated by entities and for ownership to be proved via a challenge response protocol where an entity is asked to sign a newly generated random number.

Ephemeral Recipient Identities

Ephemeral Recipient IDs in the network will be generated from a Recipient ID. When a user joins the network they will register two identities, a Transmission ID and a Recipient ID. They will only communicate with nodes via their Transmission IDs and will identify themselves to users in the network via their Recipient IDs. From a

recipient ID, an Ephemeral recipient ID will be generated through the use of network identity size *s*, rotating at a set period *p*, with a phase offset *o*.

A recipient ID will be calculated as:

$$Ephemeral\ Recipient\ ID\ =\ hash(hash(RecipientID), rotationSalt)[0: s]$$

The phase offset *o* will be:

$$o\ = hash(RecipientID)\%numOffsets$$

Where num offsets is a predefined network constant describing the number of different change points. A recommended value for a *p* of 1 day is 2^16.

The rotation salt will be dependent on the current timestamp:

The phase of the timestamp will be computed at:

$$Timestamp\ Phase\ =\ timestamp\%p$$

If the Timestamp Phase is less than phase offset o, then the rotation salt used will be:

$$Rotation\ Salt\ =\ floor((timestamp\ -\ p)/p)$$

otherwise

$$Rotation\ Salt\ =\ floor(timestamp/p)$$

Will be used.

A double hash of the RecipientID is used so the intermediary hash can be provided to third parties to track when you collide without providing them the information needed to evaluate Identity Fingerprints (see below). This can support notification systems with comparatively weak privacy properties.

Ephemeral ID Structure

The maximum size that Ephemeral IDs can grow to is 2^{64}, giving support for up to 2^{64}-1 simultaneous users. Within messages, the entire 64 bit space will be used in the message structure, with the unused bits being filled with random data. When gossiped or stored, the unused bits will all be 0.

Identity Fingerprints

Because messages will be received from other users, an easy mechanism for determining which user a message is intended for will simplify the process of filtering incoming messages.

Due to the fact that the recipient ID is known by parties the client is communicating with, they can use that as a secret in a fingerprint:

$$Identity\ Fingerprint\ = hash(Encrypted\ Message\ Payload,\ RecipientID)$$

Appendix

Group Open Optimizations

The solution to ensuring the last node cannot hide the attack in cMix paper requires *n* × *b* (*number of nodes* × *batch size*) real time exponentiations to perform. This can instead be optimized to require no real time exponentiations by adding an extra unpermuted phase of cmix and sharing the keys outright with the last node after they commit to the intermediary output. This can be further optimized by generating all keys from a seed and only sharing the seed. Furthermore, due to the fact that every node is doing the operation, the number of real time multiplication can be reduced by only adding factors to some slots, say *1/ok an*,